Python Conditionals

Main Points

1 Introducing Decision Making to Scripts

Simple script files are just lists of commands that are executed sequentially from top to bottom. Quite often it is necessary for a script to be smarter by either by making decisions based on values assigned to variables that change the flow or executing a group of commands more than once. This document focuses on decision making. The decision making process is one of the primary differences between simple and complex scripts.

This document introduces the following operators and statements used in decison making…

- Comparison operators
- Logical operators
- Branching statement groups
  - `if`
  - `if-else`
  - `if-elif-else`

2 Comparison Operators

- Making decisions requires that values and/or variables be compared
- Comparison operators require two values
  - Numeric values (integers an floats) are very common
  - Strings and other objects may also used
- Comparisons result is Boolean `True` or `False` values (not strings)

The following table summarizes *Python's* comparison operators and includes an example for each.

Operator	Description	Example	Result
`<`	Is the left side value less than the right?	`3 < 5`	`True`
`>`	Is the left side value greater than the right?	`3 > 5`	`False`
`<=`	Is the left value less than or equal to the right?	`2 <= 4/2`	`True`
`>=`	Is the left value greater than or equal to the right?	`4**2 >= 15`	`True`
`==`	Are the two values equal to each other?	`2 + 2 == 2*2`	`True`
`!=`	Are the two values not equal to each other?	`16**(1/2) != 4`	`False`
`is`	Is one object the same as another (do not use for equality ==)?	`2.0 is 2`	`False`
`is not`	Is one object not the same as the other?	`2.0 is  not 2`	`True`
`in`	Is one object in (a sub-set of) the other?	`'e' in 'yes'`	`True`
`not in`	Is one object not in the other?	`'i' not in 'team'`	`True`

Comparison operator examples

```
>>> 2 > 10
False
>>> 10 > 2
True
>>> 5 < 10
True
>>> 5 != 10
True
>>> a = 12
>>> b = 12.0
>>> a == b
True
>>> a is b
False
```

- Multiple comparisons can be used in the same expression
- `True` acts like a `1` and `False` acts like a `0` when included in math operations
- Any non-zero value is logically `True` and only zero is logically `False`
- Mathematical operators take precedence over comparison operators
- Comparison operators all have the same precedence and are evaluated left to right
- Use parentheses to group portions of expressions and change the order of execution
- When using the `in` and `not in` operators, the second object must be an iterable
  - Strings are iterables; `'we' in 'awesome'` evaluates as `True`
  - Lists, tuples, and ranges are also iterables (each of which will be introduced later)

Performing math with comparison operators

```
>>> (42 > 24) + (10 < 5)     # True + False = 1 + 0
1

>>> a = 6 < 10       # True
>>> b = 7 > 8        # False
>>> c = 5*3 == 60/4  # True
>>> a + b + c        # True + False + True = 1 + 0 + 1
2

>>> # same as above without using names
>>> (6 < 10) + (7 > 8) + (5*3 == 60/4)
2
```

Comparison operators with aphabetic strings

```
>>> # the end of the alphabet is greater than the beginning
>>> 'z' > 'a'
True
>>> 'B' < 'C'
True
>>> # all lowercase letters are greater than all uppercase letters
>>> 'A' > 'a'
False
>>> 'a' > 'A'
True
>>> 'z' > 'A'
True
>>> # what about larger strings?
>>> 'Python' > 'Java'
True
>>> 'Python' > 'C'
True
>>> 'Mechanical' > 'Electrical'
True
```

Using `in` and `not in`

```
>>> 'h' in 'Python'
True
>>> 'I' not in 'team'
True
```

3 Boolean Logic Operators

Logical operators are referred to as Booleans (object type is `bool`) and include the `and`, `or`, and `not` operators. The following table describes their basic usage.

Name	**Usage**	**Result**
`and`	`A and B`	`True` if both `A` and `B` are `True`; otherwise `False`
`or`	`A or B`	`True` if either `A` or `B` are `True`; otherwise `False`
`not`	`not A`	`True` if `A` is `False`; `False` if `A` is `True`

Logical operators using `True` and `False`

```
>>> True and True
True
>>> True and False
False
>>> False and False
False

>>> True or True
True
>>> True or False
True
>>> False or False
False
```

- Any non-zero value is considered to be logically `True`
- Zero is logically `False`
- Can be used with any strings, integers, and floats (plus more)
- Can be used alongside arithmetic operators
- The precedence of logical operators is `not`, `and`, and then `or`
- Logical operator precedence falls after the arithmetic and comparison operators
- Using logical operators with numeric values can generate curious results
- Using **`or`** with two numeric values results in...
  - The first non-zero value of the pair if at least one of the values is not zero
  - Zero if both values are zero
- Using **`and`** with two numeric values results in...
  - The second non-zero value of the pair if neither value is zero
  - Zero if either of the values is zero

Comparison and logical operators together

```
>>> 3 < 4 or 5 < 3     # same as True or True
True
>>> 3 < 4 and 5 < 3    # same as True and False
False
>>> 3 < 4 and 3 < 5    # same as True and True
True
```

Simple and and or operations with numeric values

```
>>> 3 and 7     # True and True => result is second value
7
>>> 10 and 0    # True and False => result is second value
0
>>> 5 or 10     # True or True => result is first value
5
>>> 0 or 7      # False or True => result is second value
7
```

Using not with numeric values

```
>>> not 25    # same as not True
False
>>> not 0     # same as not False
True
```

Complex logical operations with numeric values

```
>>> 25*((12 and 0) + (not 0) + (0 or 5))
150
>>> # not 0 is False, which is treated as 1 for math
>>> # 25 * (0 + 1 + 5)
```

Checking if a value is between two others

```
>>> x, y = -2, 5
>>> -5 < x < -1
True
>>> # the above line does the same as the line below
>>> -5 < x and x < -1
True
```

Mixing logical and comparison operators

```
>>> x, y = -2, 5
>>> not (y < 7)
False
>>> not ((y >= 8) or (x < -1))
False
>>> not (y >= 8) or (x < -1)
True
```

4 Conditional Branching

4.1 The `if` Statement Group

The `if` statement is part of a block or group of commands.

- First line contains the `if` statement plus a comparison followed by a colon
- The colon indicates that the line is the first of a group of lines/commands
- The lines that follow contain code that executes only when the comparison is `True`; must be indented (usually 4 spaces) relative to the `if` line
- Execution will skip the indented lines if the initial comparison is `False`

An `if` group where the comparison is `True`

```
x = 9
if x < 10:  # any comparison, must have the colon at the end
    # the indented code only executes if the comparison is true
    print('x is less than 10')
print('This line always executes: x =', x)
```

```
x is less than 10
This line always executes: x = 9
```

The `if` statement comparison is `False` this time

```
x = 10
if x < 10:
    print('x is less than 10')
print('This line always executes: x =', x)
```

```
This line always executes: x = 10
```

The following user-defined function prints `That is odd` only when the argument `x` is odd. It uses the remainder division (%) operator in the comparison. Nothing is printed when `x` is even.

A note on indentation in a function definition: the function body gets indented 4 spaces. The if statement line inside the function definition needs to match the function body indentation. The commands belonging to the if group must be indented an additional 4 spaces.

Function with an if statement – is_it_odd(x)

```
def is_it_odd(x):
    if x % 2:    # 0 equals False, remainder is 0 when x is even
        print("That is odd")
```

```
>>> is_it_odd(5)
That is odd
>>> is_it_odd(12)    # will not print anything
>>> is_it_odd(33)
That is odd
>>> is_it_odd(-10)   # will not print anything
>>> is_it_odd(-13)
That is odd
```

4.2 The if-else Statement Group

An if-else statement group starts with an if statement and its code and adds code that only executes when the if comparison is False. The second group of code is the else statement group.

- An else statement is added immediately after the last indented line of the if group
- The else must have the same indentation as the if line with which it is paired
- The else line can not include a comparison
- The lines following the else contain the code that executes when the if condition is False; must be indented relative to the else line

Structure of an if-else group

```
x = 9
if x < 10:  # any comparison; must have the colon at the end
    # this indented code executes if the comparison is True
    print('x is less than 10')
else:       # no comparison allowed; must have the colon at the end
    # this indented code executes if the comparison is not True (False)
    print('x is greater than or equal to 10')
print('This line always executes: x =', x)
```

The following function definition includes both if and else blocks. Note where the colons ':' and the indenting. Noitce that the else stands alone without any comparison.

Function with `if-else` – `is_single(y)`

```
def is_single(y):
    if -10 < y < 10:
        print(f'{y} is a single digit number')
    else:
        print(f'{y} is not a single digit number')
```

```
>>> is_single(5)
5 is a single digit number
>>> is_single(10)
10 is not a single digit number
>>> is_single(-11)
-11 is not a single digit number
>>> is_single(-9)
-9 is a single digit number
```

4.3 The `if-elif-else` Statement Group

The addition of one or more `elif` (short for "else if") statement groups to an `if-else` group allows for any number of possible conditions. The format of the `elif` statement is essentially the same as the `if` statement.

- All `elif` groups are located after the initial `if` group and before an `else` group
- Use of this statement group type does not mandate the use of an `else`
- Only the first condition that evaluates to `True` is executed
- Once an `if` or `elif` group is executed, all others will be skipped
- If the `if` and all `elif` statements are `False`, then the `else` group will be executed

Structure of an `if-elif-else` group

```
x = 9
if x < 10:  # any comparison; must have the colon at the end
    # this indented code only executes when the 'if' is True
    print('x is less than 10')
elif x == 10:
    # this indented code only executes when the 'elif' is True
    print('x is equal to 10')
else:
    # this indented code only executes when all comparisons are False
    print('x is greater than 10')
print('This line always executes: x =', x)
```

The code below defines a function named `diceroll()`. The function randomly generates two values from 1 to 6 (the rolls of two dice) and prints a statement based on the two values. The function includes an `if-elif-else` group with two `elif` statements.

Function with `if-elif-else` – `diceroll()`

```
def diceroll():
    import random
    # snake eyes is a roll of 2
    # a natural is a roll of 7 or 11
    # boxcars is a roll of 12
    # anything else is "nothing special"
    die1 = random.randint(1, 6)
    die2 = random.randint(1, 6)
    print('Die 1 =', die1, '\tDie 2 =', die2)
    if die1 + die2 == 2:
        print('Too bad')
        print(f'{die1} + {die2} is snake eyes')
    elif (die1 + die2 == 7) or (die1 + die2 == 11):
        print('Winner, winner, chicken dinner')
        print(f'{die1} + {die2} = {die1 + die2}, a natural')
    elif die1 + die2 == 12:
        print("It's not your day")
        print(f'{die1} + {die2} is boxcars')
    else:
        print('Better luck next time')
        print(f'{die1} + {die2} is nothing special')
```

```
>>> diceroll()
Die 1 = 1     Die 2 = 3
Better luck next time
1 + 3 is nothing special
>>> diceroll()
Die 1 = 2     Die 2 = 4
Better luck next time
2 + 4 is nothing special
>>> diceroll()
Die 1 = 5     Die 2 = 5
Better luck next time
5 + 5 is nothing special
>>> diceroll()
Die 1 = 3     Die 2 = 6
Better luck next time
3 + 6 is nothing special
```

5 The Ternary Operator; A *Pythonic* Technique

Python allows for an assignment and a simple `if-else` in single line of code. This operation is called the *ternary operator*. It does the same thing in one expression that a standard `if-else` does with four.

```
var_1 = value_1 if <<comparison>> else value_2
```

- `var_1` will be assigned `value_1` if the comparison is `True`
- Otherwise, `var_1` will be assigned `value_2`

A simple `if-else` group

```
y = 10
if y > 10:
    x = 25
else:
    x = 20
print('x =', x)
```

Using a ternary operator instead

```
>>> y = 10
>>> x = 25 if y > 10 else 20
>>> print('x =', x)
x = 20

>>> y = 12
>>> x = 25 if y > 10 else 20
>>> print('x =', x)
x = 25
```

6 Some Creative Commons Reference Sources for This Material

- *Think Python 2nd Edition*, Allen Downey, chapter 5
- *The Coder's Apprentice*, Pieter Spronck, chapter 6
- *A Practical Introduction to Python Programming*, Brian Heinold, chapter 4
- *Algorithmic Problem Solving with Python*, John Schneider, Shira Broschat, and Jess Dahmen, chapter 11